Raymond Lin
CS M146 - Introduction to Machine Learning
304937942

Problem Set 4

1. PCA and Image Reconstruction

a. image at index 0

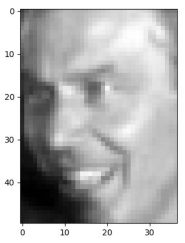


image at index 100

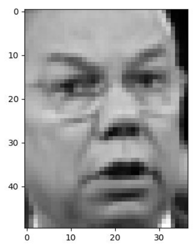
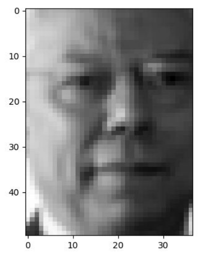
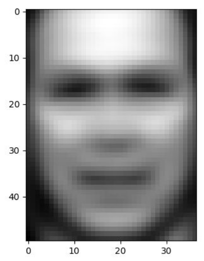


image at index 1000

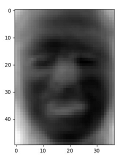


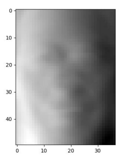
the “average” image

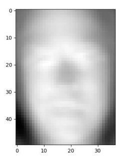


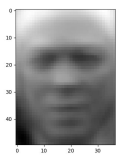
The average face looks kind of spooky! It looks like there are no eyes and that the facial features are more blurry than those of the other images. This is to be expected, because we are taking an average of all the other images. We can see on average what value a particular pixel in the picture takes, or in other words, how likely the pixel is to be a lighter or darker shade.

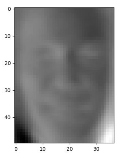
b. Eigenfaces

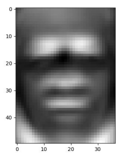

1

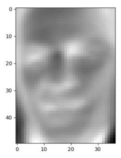

2

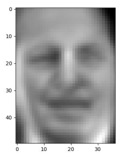

3

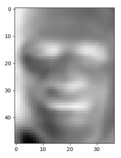

4

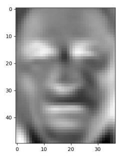

5

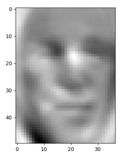

6

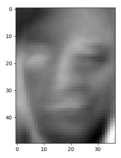

7


8


9

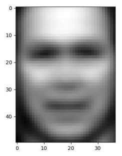

10

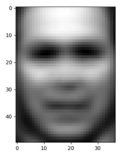

11

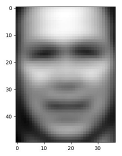

12

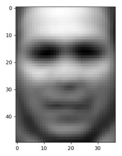
These faces are selected as the top eigenfaces, because they capture the most information in the face. This is because each of them comes from a primary component of the face image, and thus it captures the axis with the highest variance.

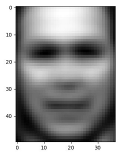
c. l = 1

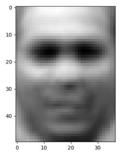

1

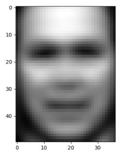

2

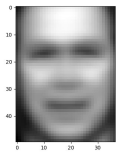

3

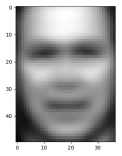

4

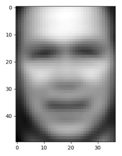

5

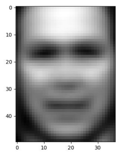

6

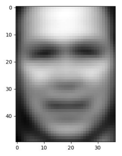

7


8

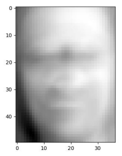

9

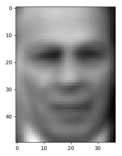

10

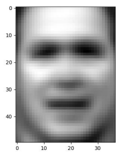

11

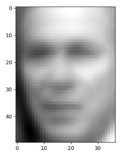

12

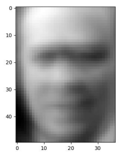
l = 10

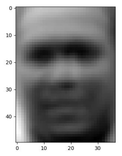

1

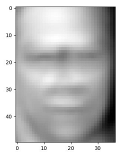

2

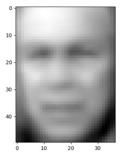

3

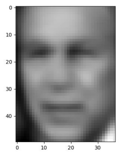

4

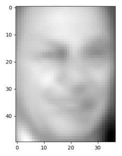

5

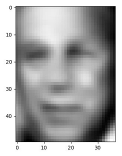

6

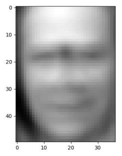

7


8

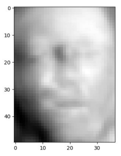

9

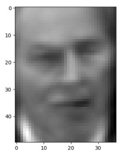

10

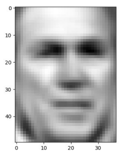

11

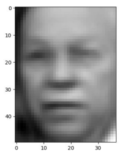

12

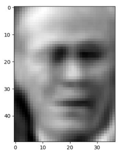
l = 50

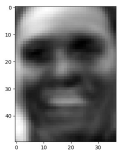

1

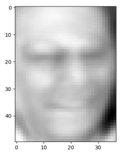

2

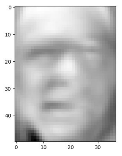

3

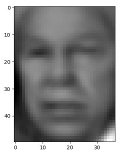

4

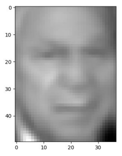

5

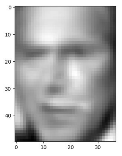

6

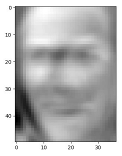

7


8

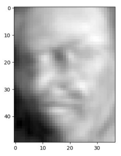

9

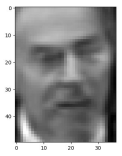

10

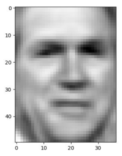

11

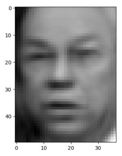

12

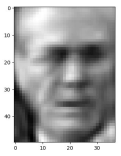
l = 100

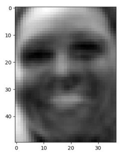

1

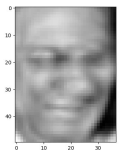

2

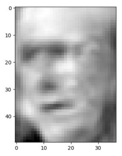

3

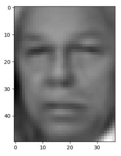

4

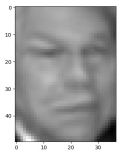

5

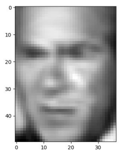

6

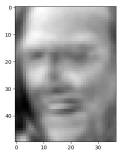

7


8

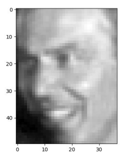

9

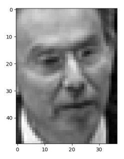

10

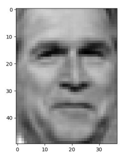

11

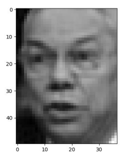

12

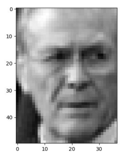
l = 500

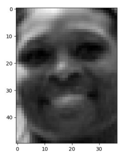

1

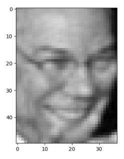

2

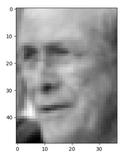

3

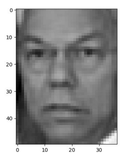

4

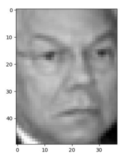

5

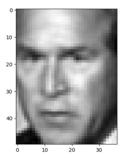

6

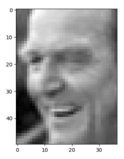

7


8

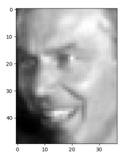

9

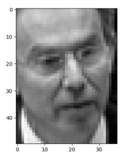

10

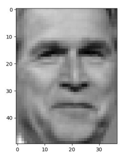

11

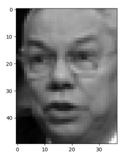

12

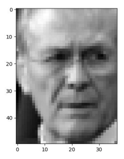
l = 1288

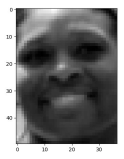

1

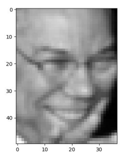

2

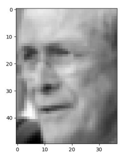

3

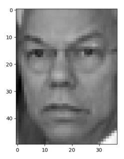

4

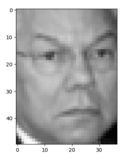

5

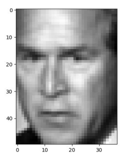

6

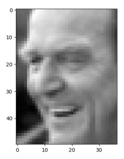

7


8


9


10


11


12

The ‘l’ value represents the number of principal components to use to reconstruct the image. As we increase the number of principal components, the reconstructed image becomes closer and closer to the original. Using this strategy, we can identify which features are extraneous and which features are important. We can see that up to a certain ‘l’-value, the reconstruction does not improve much, such as between 500 and 1288.

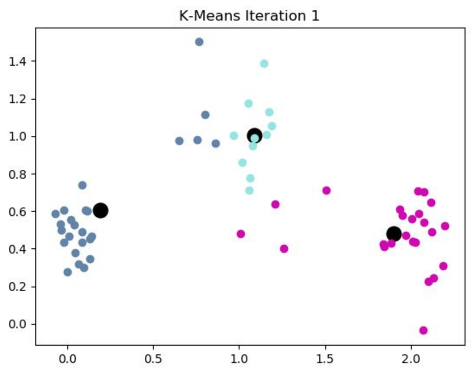
2. K-Means and K-Medoids

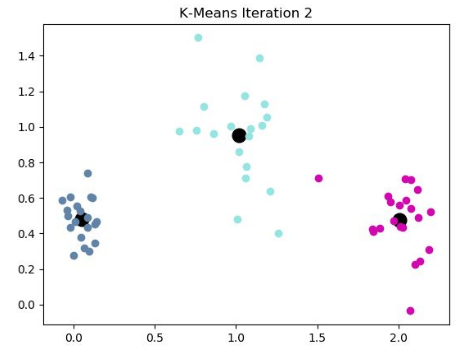
a. If we try to minimize $J(c, \mu, k)$ over the parameters, including k, we get a value of 0. However, because we can allow k to change, we can set it to be equal to the number of data points. Thus, each data point is in its own cluster, and each cluster’s mean is the same as the only data point that exists inside it. In other words, k is the number of data points, μ is the same as the single data point in the cluster, and c is the number of the data point (ie. if we have x_1, then the associated c is also 1). As a result, we get the minimum value of the objective function to be 0. This is bad, because we aren’t actually clustering anything, if each data point gets its own cluster.

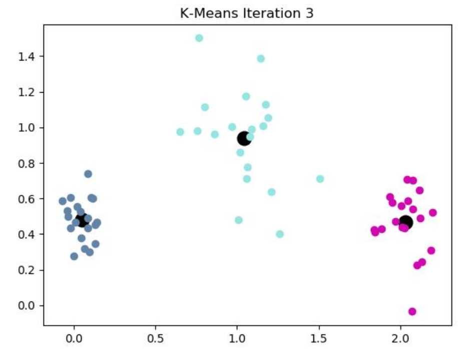
b. done

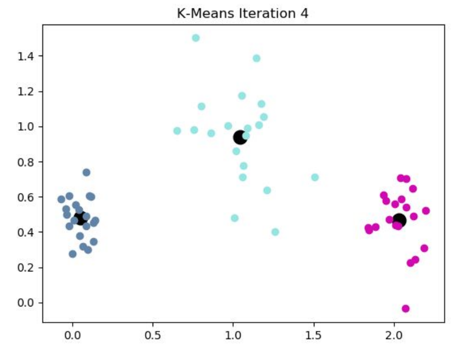
c. done

d. Using K-Means and random_init

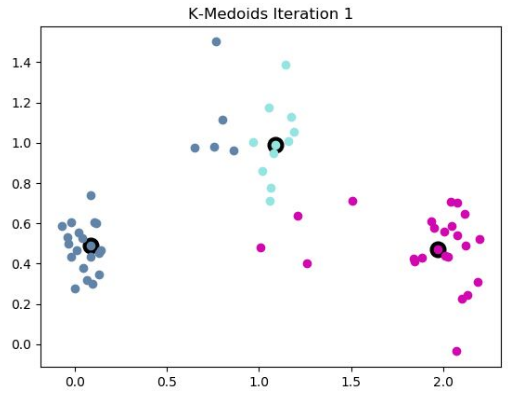


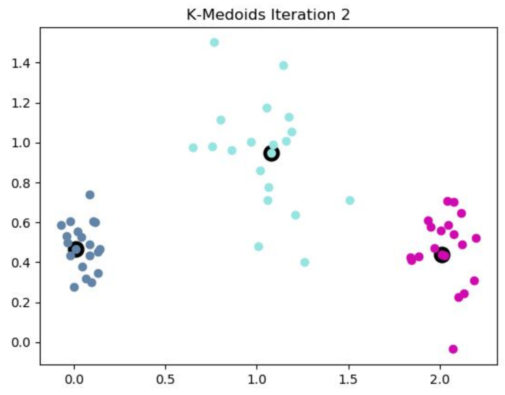

K-Means Iteration 2
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0

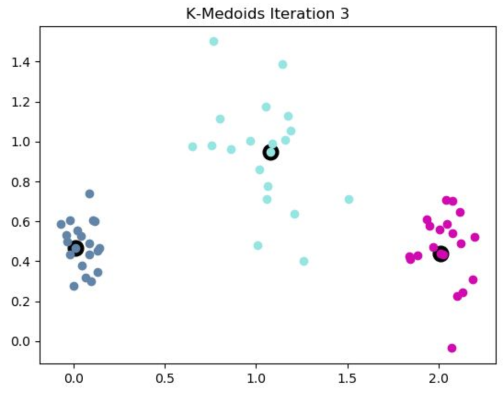

K-Means Iteration 3
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0


K-Means Iteration 4
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0

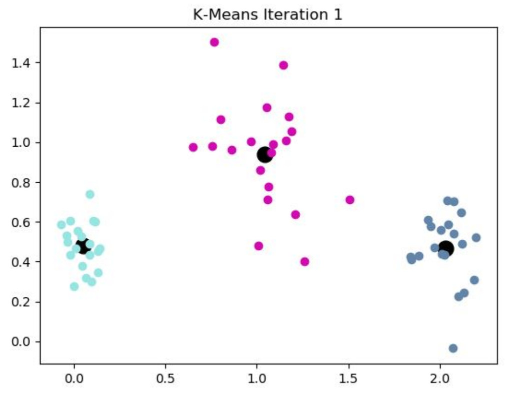
e. Using K-Medoids and random_init

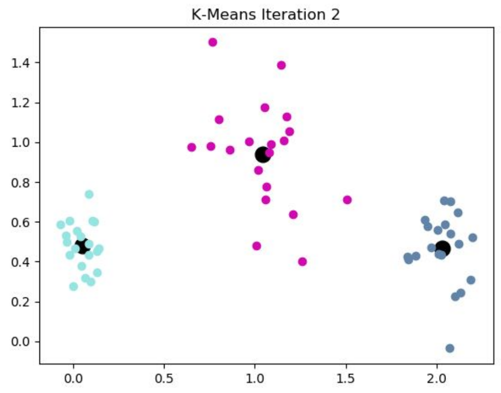




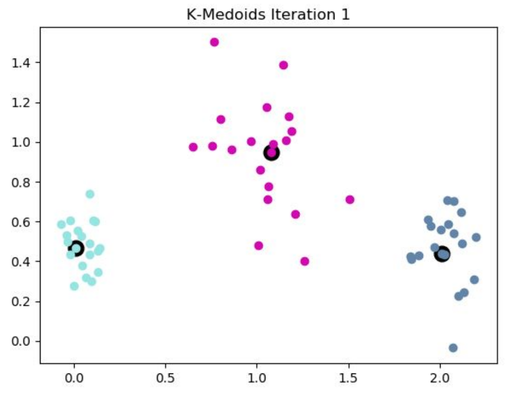


f. Using K-Means and cheat_init





Using K-Medoids and cheat_init



Using ‘cheat_init’, we see that the centers of each cluster are already at the required positions when the algorithm starts running. As a result, we require much fewer iterations for the algorithm to converge.

3. Clustering Faces

a. Scores

	Average	**Min**	**Max**
K-Means	0.623	0.575	0.775
K-Medoids	0.628	0.575	0.713

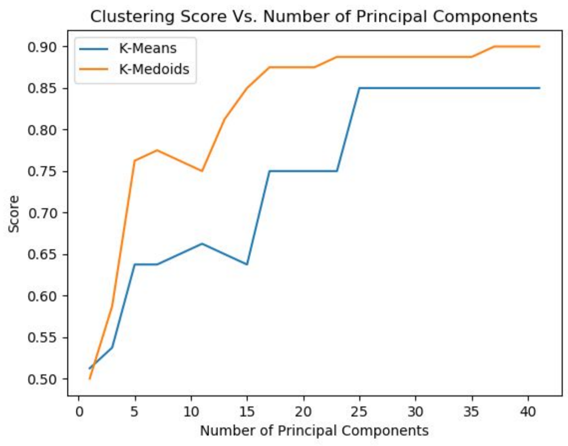
On average, both clustering methods perform relatively similarly. They both have a similar average and minimum score. However, the K-Means algorithm has a higher maximum score than that of K-Medoids. This is expected, because the center for K-Medoids is restricted and limited to existing data points, while K-Means is free to choose an imaginary data point that is closest to all data points.

Runtime (s)

	Average	**Min**	**Max**
K-Means	0.077	0.045	0.113
K-Medoids	0.196	0.125	0.333

It seems like K-Means runs faster than K-Medoids, by a factor of around 3. This is also to be expected, because K-Medoids must compute the distance from each data point to every other data point, which is approximately a time complexity of O(N^2). In contrast, K-Means simply has to take an average of the feature values of all of the data points. This has a time complexity of around O(N).

b.

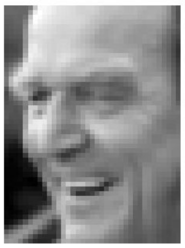


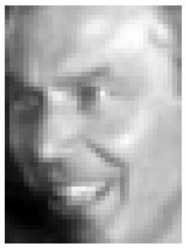
As the number of principal components increases, so does the clustering score. This is expected, because as we use more principal components, the image begins to look more and more like the original. As a result, the clustering algorithms can do a better job of grouping like images together. We also see that the K-Medoids algorithm performs slightly better than K-Means. This is probably because K-Medoids does not let outliers skew its center point, as it is a medoid, while K-Means does, as it is an average.

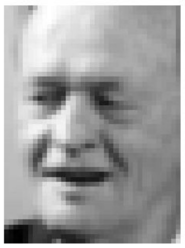
c. First, I found the number of unique labels in the dataset. I found that there are 19 unique labels, which were numbered from 0 to 18. Then, for each unique combination of two labels in the dataset, I extracted 40 images from the dataset that were associated with each label (total of 80 images). I then ran a clustering algorithm on the extracted dataset, with k = 2, since there are only two clusters. I picked the K-Medoids clustering algorithm, because from the previous problem, I found that it scored higher than K-Means. Then, I computed the score from the clustering algorithm. The pair of images with the best score corresponds to the two most discriminative images, while the pair of images with the worst score corresponds to the two least discriminative images.

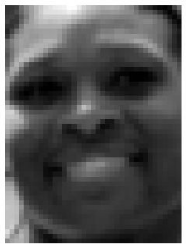
Least Discriminative

Most Discriminative









5 17 9 16

For the most discriminative pair, we received a score of 0.988, which means that the algorithm was able to cluster images 9 and 16 vert well. For the least discriminative pair, we received a score of 0.500, which means that the algorithm performed poorly when trying to cluster images 5 and 17. Visually, we can see that our results imply that our methodology and algorithm implementation is correct. The least discriminative images look quite similar, while the most discriminative images look very different.